"Hey — it works!"

Jeremy Gibbons
University of Auckland
jeremy@cs.aukuni.ac.nz

G'day!, as they say down here, and welcome to *"Hey — it works!"*. This column is designed for little tricks — even ones that seem trivial or inelegant — that others may find useful. If you have any such items, please send them in to me at the address above. I am particularly looking for short items (up to about a page), although in this issue we have space for a few longer ones. These are: a piece from Donald Arseneau describing a smart macro for emphasis, which automatically adds italic corrections where they are necessary and doesn't where they are not; one from Ed Baker about a little TEX utility for formatting address labels interactively; and one from Fred Lauwers describing a macro for making dropped capitals.

But just before we start... Further to David Lau's piece on striking through text in the previous issue, Alan Hoenig `<ajhjj@cunyvm.cuny.edu>` observes that '*the* right way to strike through text [is to] use virtual fonts'; this approach works for text of arbitrary length, too, and also works for underlining.

Italic correction everywhere

Donald Arseneau
Tri-University Meson Facility
asnd@erich.triumf.ca

Many people are not aware of, or simply ignore, the 'italic correction' command (`\/`) that should be inserted after slanted text for proper spacing (see The TEXbook, p. 14). Compare d*d*d with d*d*d. It is a tedious and thankless task inserting such corrections in collaborators' flawed LATEX input. Instead, it would be nice to have LATEX insert the corrections automatically. A problem that arises is that the `\/` should be omitted when followed by low punctuation (period or comma). Furthermore, when the surrounding text is italic, `\em`phasis produces *roman* text, and the italic correction should be placed *before* it. This is a particular problem for macros like `\newcommand{\abi}{{\em ab~initio\/}}` that do not know their context. Here is a replacement for LATEX's `\em` command that correctly inserts its own italic correction in these cases so you can avoid typing `\/` altogether.

First, modify the `\pem` command slightly so that it inserts a 'smart' italic correction; either immediately if the text switches to roman, or launching it towards the end of the group if the text switches to italics.

```
\def\pem{\ifdim \fontdimen 1\font>0pt \rm
   \else \it \expandafter\aftergroup \fi \smartitcor}
```

or, for NFSS,

```
\def\pem{\@nomath\em \ifdim\fontdimen\@ne\font>\z@ \normalshape
   \else \it \expandafter\aftergroup \fi \smartitcor}
```

The smart italic correction is only executed in horizontal (rl) mode. When it is executed, it first looks at the next character (token) to see if it is a comma or period (or something else that is short).

```
\def\smartitcor{\ifhmode \expandafter \itpunclook \fi}
\def\itpunclook{\begingroup \futurelet \ITCtempa \itcortest}

\def\itcortest{%                  test next character (\ITCtempa)
   \def\ITCtempb{\ITCtempa}%      hide next when skipping
   \ifcat\noexpand\ITCtempa ,%    if it could be punctuation,...
      \setbox0\hbox{\ITCtempb}%   check its size...
      \ifdim\ht0<.3ex %           and if it is short,...
         \let\itcordo\endgroup %  suppress italic correction
   \fi\fi \itcordo}%              otherwise, do it
```

The `\noexpand\ITCtempa` is used in case the next token is a macro, which should not be expanded out of context. The extra level of definition, `\ITCtempb`, is used in case the next token is a conditional command like `\if` or `\fi` which could not be skipped over properly when the `\ifcat` is false.

Finally, the italic correction must *precede* any spaces or it is useless.

```
\def\itcordo{% Put \/ before any preceding space.
   \skip0=\lastskip
   \ifdim\skip0=0pt\/\else\unskip \/\hskip\skip0 \fi
   \endgroup}
```

Here we check for preceding space (glue or skip) by comparing the length of `\lastskip` with `0pt`. If the previous glue was `\hfil` there will be no italic correction, but it is probably unnecessary in that case anyway.

These macros are intended for `\em` delimited by grouping, such as

```
My book is called {\em You named the baby {\em what?!}}\\
\begin{em}
   Sam Spade had {\em never} seen that {\em modus operandi} before.
\end{em}
```

My book is called *You named the baby* what?!
Sam Spade had never *seen that* modus operandi *before.*

There might be problems if you use `\em` improperly as an on/off switch or if you rely on implicit grouping; *e.g.*, if you use it in a save-box (`\setbox`) or in a primitive `\halign` without extra braces (although a tabular environment works fine). Even when it fails, the italic correction usually just disappears harmlessly. And hey, it works for the 99 44/100% of the time that you emphasize `{\em this way}`.

Plain TEX, Self-Prompting, Label Maker

Ed Baker
EBTS, Norfolk, MA
ejb@world.std.com

Here's a little, self prompting, plain TEX, address label maker that I use for making self-adhesive address labels. Each label is 4 in by 3 in, and the top half is pre-printed with the company logo and return address.

The TEX file creates a label of the following format:

First-Name Mid-Init Last-Name
Company-Name
Address-1
Address-2
City, ST ZIP

Here are some of the features:

- If any prompt is answered with ⟨cr⟩, nothing will be typeset. That is, no blank lines will appear in the address label.
- Labels are printed with the `cmssbx10` font at 12 pt. The Post Office prefers sans-serif fonts.
- `\baselineskip` is set to 15 pt to increase legibility.
- `\voffset` can be adjusted to offset the vertical placement of the address origin. (This example doesn't use `\hoffset`, but it could be added easily if you need it.)

Although this example uses a USA addressing format, you can easily adapt it to any country.

The tricky part is getting TEX to do nothing if you simply press carriage return as an answer to a prompt. Here's the code segment that does the trick:

```
\setbox0=\hbox{\PersonName} \ifdim\wd0=0pt\else\line{\box0\hfill}\fi
```

Notice that the 'true' action is null (There is nothing between `\wd=0pt` and the `\else` action).

The answer to the PersonName prompt is stored in box 0. The width of box 0 will be non-zero if you entered any characters. However, the width will be 0 pt if you simply press carriage return. Next, the `\ifdim` command queries the width of box 0. If the answer is 0 pt, nothing happens. Otherwise box 0 is typeset as an address line.

It's small, but useful.

I'd like to thank Dr. Richard Kinch, Ithaca, NY for the inspiration to create this utility.

```
%%======================================== Address Label Generator
\font\cmssbxii = cmssbx10 at 12pt
\obeylines\cmssbxii
\baselineskip = 15pt
\noindent
\voffset = 1in
\immediate\write16{This Program will prepare an EBTS label for you}
\immediate\write16{It will ask for the following information:}
\immediate\write16{      First-Name MI Last-Name}
\immediate\write16{      Company Name}
\immediate\write16{      First Address Line}
\immediate\write16{      Second Address Line}
\immediate\write16{      City, ST ZIP}
\immediate\write16{Just follow the prompts}
\immediate\write16{Enter First-Name Middle-init Last-Name}
   \read-1to\PersonName
\immediate\write16{Enter Company Name}
   \read-1to\CompanyName
\immediate\write16{Enter Address Line 1 of 2}
   \read-1to\AddressLineOne
\immediate\write16{Enter Address Line 2 of 2}
\immediate\write16{Press Enter if Blank}
   \read-1to\AddressLineTwo
\immediate\write16{Enter City, ST ZIP}
   \read-1to\CityStateZipName
\setbox0=\hbox{\PersonName}%
   \ifdim\wd0=0pt\else\line{\box0\hfill}\fi
\setbox0=\hbox{\CompanyName}%
   \ifdim\wd0=0pt\else\line{\box0\hfill}\fi
\setbox0=\hbox{\AddressLineOne}%
   \ifdim\wd0=0pt\else\line{\box0\hfill}\fi
\setbox0=\hbox{\AddressLineTwo}%
   \ifdim\wd0=0pt\else\line{\box0\hfill}\fi
\hbox{\CityStateZipName}
\vfill
\eject
```

```
\bye
```

A style for dropped capitals

Fred J. Lauwers (FJL)
Technische Universiteit Delft
fred@dutspf.tudelft.nl

Peter Flynn dedicates a subsection of *Typographer's Inn* (TTN 1,4) to the problem of dropped capitals. This is my answer to some of his questions. Basically, I don't start with the capital, but I let the capital's size depend on the chosen design. I wrote a dropped caps macro for use with LaTeX, and took the following design decisions.

- First, let the writer choose her favourite design, within certain limitations though, and let the macro do the dirty work: adjusting the cap to the optimal size.
- Let the writer decide how many lines she wants to dropindent. Let her also decide in which font the cap should appear, whether to start the cap *at* the left margin, or somewhat to the left (into the margin) or to the right (into the text). Let her decide what she puts into dropped caps: a single character, or even one or more whole words, or without caps at all.
- The limitations: Let the cap's baseline be aligned with the baseline of the lowest of the indenting lines. Let the cap's top line be aligned with the top line of the first of the indenting lines. Don't let the writer mess with this!

Usage My `dropcaps.sty` is a LaTeX style file, that defines a macro command `\bigdrop` that enables you to drop capitals at the beginning of a paragraph. The layout can be controlled in various ways, with four parameters (not optional). For instance, the first paragraph started with

```
\bigdrop{-2pt}{2}{cmr10}{P}eter ...
```

The required parameters are:

1. a *length*, that denotes the distance from the left margin (to the left or to the right) at which the dropped capital starts, in this case 2pt into the left margin: `-2pt`.
2. an integer value, that denotes the number of lines to be indented, in this case 2 lines.
3. the font to be used for the dropped capital, in this case `cmr10`.
4. the character or text to be dropped, in this case a single `P`.

The messy details When the macro `\bigdrop` is called (with four parameters), this is what happens. First, a capital 'I' is typeset in the current text font and put in a box (`box1`). The desired font height (`\wantedheight`) is then computed from $(n-1) \times$ `\baselineskip` $+$ `\ht1`, where n is the number of lines to indent. Then the procedure differs, depending on whether fonts at arbitrary sizes are available.

with arbitrary sizes: In this case the text to be capitalized, typeset in the chosen font (at scale `1000`), is put in a box (`box0`), with height `\actualheight` = `\ht0`. Now, the scaling factor for this font can be computed from dividing `\wantedheight` by `\actualheight` and multiplying this by `1000`. Because divisions are integer, the heights are first put in count registers, and then, to prevent from rounding errors and overflow, `\wantedheight` $\times$ 10 is divided by `\actualheight`/100. The outcome of this is the scale at which the chosen font is typeset.

without arbitrary sizes: Without the freedom to use arbitrary sizes, things are less elegant, for only the standard magsteps—half, and zero to five—are available for scaling fonts. The text to be capitalized in the chosen font, is first typeset at `\magstep0` and put in a box (`box0`), with height `\ht0`.

If `\wantedheight` is bigger than `\ht0`, the text is typeset at `\magstephalf`, and put in `box0` again. If `\wantedheight` is still bigger than `\ht0`, again the `\magstep` is increased one step, et cetera. This process is repeated until `\wantedheight` is smaller than `\ht0`.

The result of the computation can be observed on the screen and in the `.log` file. Each time a capital is dropped, two lines like

```
*** Needed height: 18.83331pt
*** cmr10 scaled 2488 ==> height: 17.0013pt (error: 1.83202pt).
```

appear. This example is from processing this document with $n = 2$ and font `cmr10`, without using fonts at arbitrary sizes.

At the end of the computation `box0` contains the text, typeset in the desired font and scaled such that `\ht0` equals the height of the indenting lines. Then, `\hangindent` and `\hangafter` are set according to `box0` and the parameter values: `\hangindent` = `\wd0` + `#1`, and `\hangafter` = −`#2`. Precautions are taken—an extra line is indented—for the case that `\dp0` is not equal to zero. Finally, in a LATEX picture environment, the text is placed where we want it.

Requirements In order to use this style file, a big version of LATEX is needed, because almost always fonts have to be generated, and already from 4 indenting lines, they tend to become very large. I tested this with emTEX's TEX386 (with emx) on a 486DX33 PC, and with emTEX's bTEX186 on a AT286 PC.

Also desirable is a version of DVIPS by Tomas Rokicki with `MakeTeXPK` running, so the generation of fonts will be done automatically. For me the recently uploaded DVIPS 5.518 works great on both machines.

Alternatives Only one style file that does something similar is listed in TEX-index v1.06 by David M. Jones, namely `drop.sty` by David G. Cantor. The latest version is dated 17/02/1988. This style file also defines a macro to drop capitals, but has no possibilities to accept controlling parameters, except for the choice of font. You are not able to choose the desired height by supplying the number of lines to be indented.

The dropped capital has a fixed size, namely `\magstep5`, and always two lines are indented. This font height is somewhat smaller than the height of two indenting lines, so the vertical placement of the dropped capital's baseline lies not at the same height as the vertical placement of the baseline of the second indenting line (next to it). The advantage of this is that everyone has this font.

```
\def\fileversion{0.30}
\def\filedate{13 Sep 1993} \def\docdate{\filedate}
\typeout{\space\space Style option: 'dropcaps' v\fileversion
   \space\space <\filedate> \space\space (FJL)}%
\def\yes{y }\def\no{n }\newif\ifarbsizes
   \message{*** Are arbitrary font sizes available [y/n] ? }
   \read-1 to \answer
   \ifx\answer\yes\arbsizestrue\else\arbsizesfalse\fi
\newcount\bigscale
\ifarbsizes % \typeout{*** Good choice.}%
   \newcount\wantedcount\newcount\actualcount%
   \newdimen\actualheight%
   \def\findsize#1#2{%
      \bigscale=1000 \font\BIG=#1 scaled \bigscale%
      \setbox0=\hbox{\BIG #2\/}%
      \actualheight=\ht0%
      \typeout{*** Needed height: \the\wantedheight,
         \space Unscaled height: \the\actualheight}%
      \wantedcount=\wantedheight \actualcount=\actualheight%
      \advance\actualcount by50 \divide\actualcount by100%
      \multiply\wantedcount by10 \divide\wantedcount by\actualcount%
         % ie \wantedcount:=1000*\wantedcount / \actualcount
      \bigscale=\wantedcount \font\BIG=#1 scaled \bigscale%
      \setbox0=\hbox{\BIG #2\/}}
\else % \typeout{*** OK, prepare for bad output...}%
   \def\nextmagstep#1#2#3{\bigscale=#3\font\BIG=#1 scaled\bigscale%
      \setbox0=\hbox{\BIG #2\/}}%
```

```
   \def\findsize#1#2{%
      \typeout{***  Needed height: \the\wantedheight}%
      \nextmagstep{#1}{#2}{\magstep0}%
      \ifdim\ht0<\wantedheight\nextmagstep{#1}{#2}{\magstephalf}%
      \ifdim\ht0<\wantedheight\nextmagstep{#1}{#2}{\magstep1}%
      \ifdim\ht0<\wantedheight\nextmagstep{#1}{#2}{\magstep2}%
      \ifdim\ht0<\wantedheight\nextmagstep{#1}{#2}{\magstep3}%
      \ifdim\ht0<\wantedheight\nextmagstep{#1}{#2}{\magstep4}%
      \ifdim\ht0<\wantedheight\nextmagstep{#1}{#2}{\magstep5}%
      \fi\fi\fi\fi\fi\fi}
\fi
\newcount\down \newdimen\saveunitlength \newdimen\wantedheight
\def\bigdrop#1#2#3#4{%
   \saveunitlength=\unitlength \unitlength=\baselineskip%
   \setbox1=\hbox{\the\font I}%
   \wantedheight=#2\baselineskip%
   \advance\wantedheight by -\baselineskip%
   \advance\wantedheight by \ht1%
   \findsize{#3}{#4}%
   \advance\wantedheight by -\ht0%            only for typeout...
   \typeout{***  #3 scaled \the\bigscale \space ==> height:
      \the\ht0 \space (error: \the\wantedheight).}%
   \hangindent=\wd0 \advance\hangindent by #1%
   \hangafter=-#2%
   \ifdim\dp0>0.25\baselineskip%            0pt ?
      \typeout{*** extra line(s) indented: non-zero depth: \the\dp0}%
      \loop\ifdim\dp0>\baselineskip
         \advance\baselineskip by \baselineskip
         \advance\hangafter by -1
      \repeat
      \advance\hangafter by -1
      \baselineskip=\unitlength
   \fi%
   \noindent%
   \down=-#2 \advance\down by 1%
   \begin{picture}(0,0)%
      \put(0,\down){\makebox(0,0)[br]{\box0}}%
   \end{picture}%
   \unitlength=\saveunitlength}
```